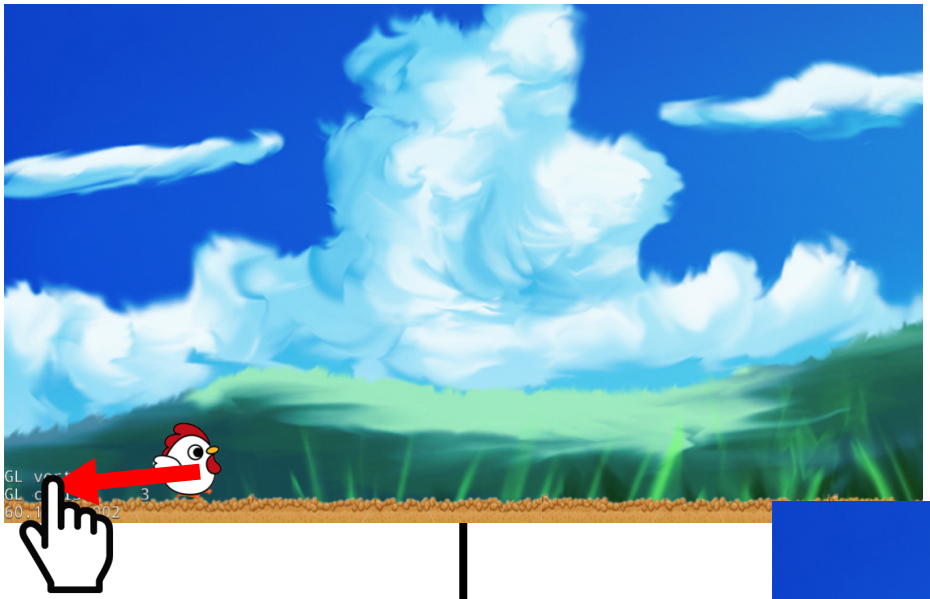
Cocos2d-xで作る物理演算ゲーム

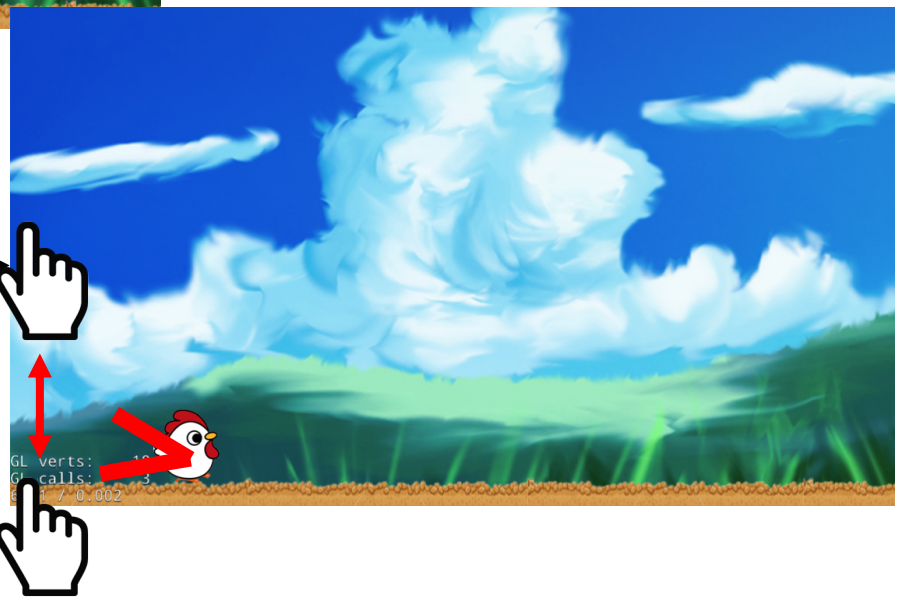
やり直し処理を入れてみる編

= 2016年2月20日 =

前回のあらすじ



チキンにタッチして大きく引っ張っても…



それ以上は伸びない！
さらに上下に指を移動させると長さをキープしたまま線分の角度が変わる！

ここまでのソースプログラムはここ

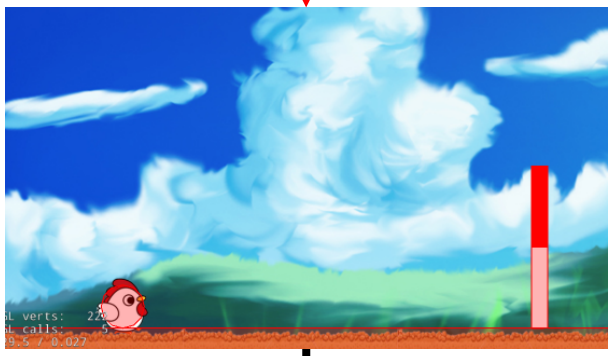
http://monolizm.com/sab/src/AngryChicken14.zip

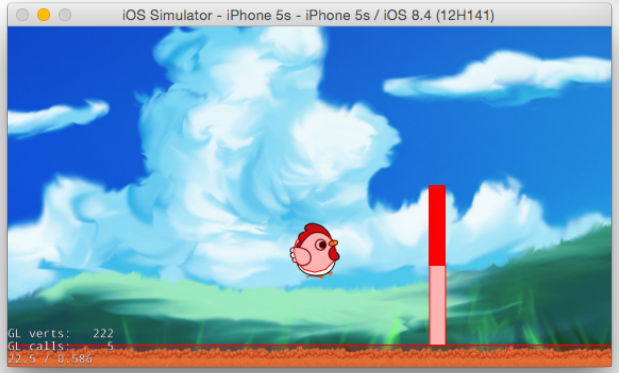
GETだぜ！

今回はやり直し処理を
入れてみます。

今回はこちら

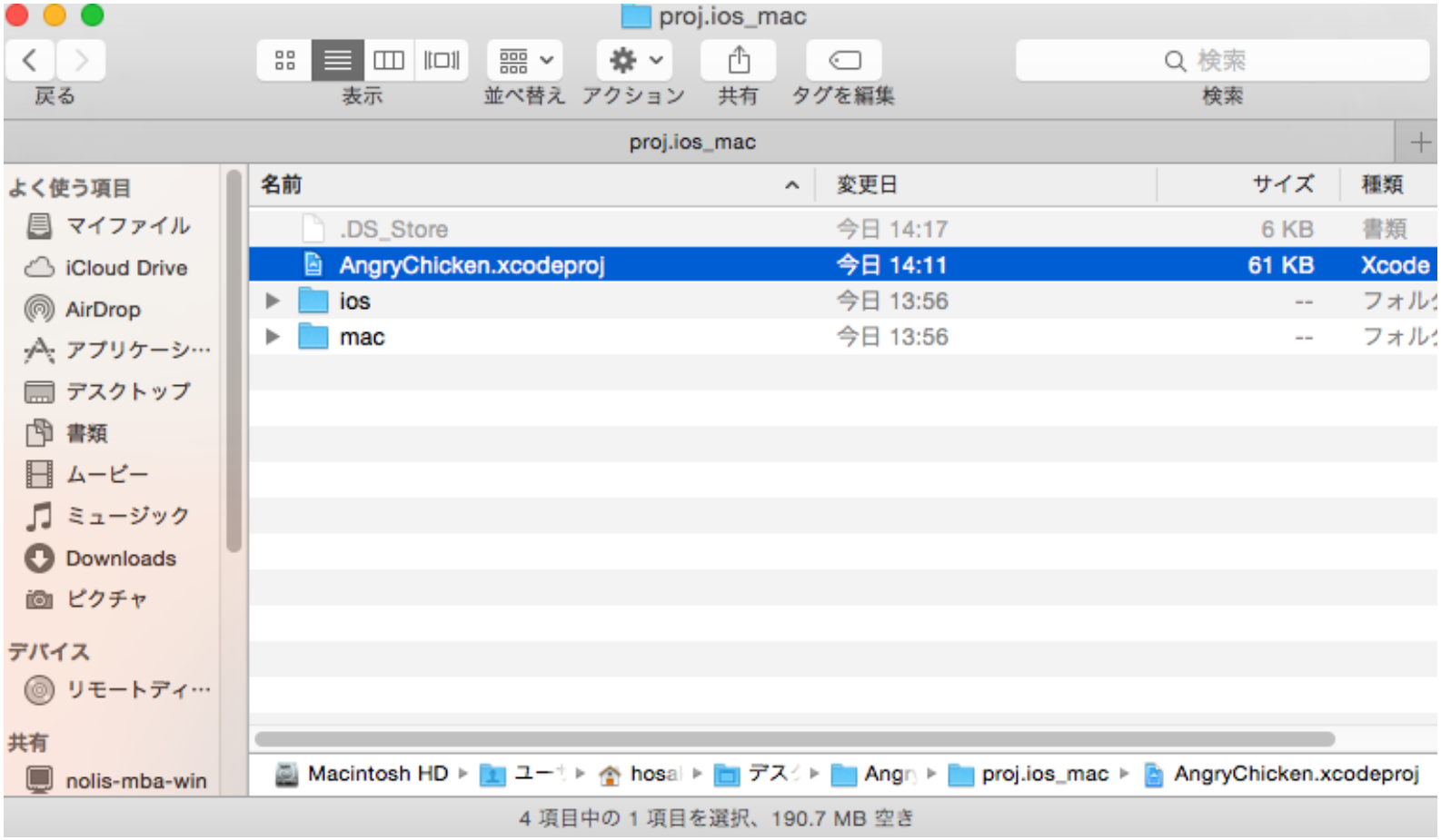
静止したら
自動的に初期化





まずは起動しよう

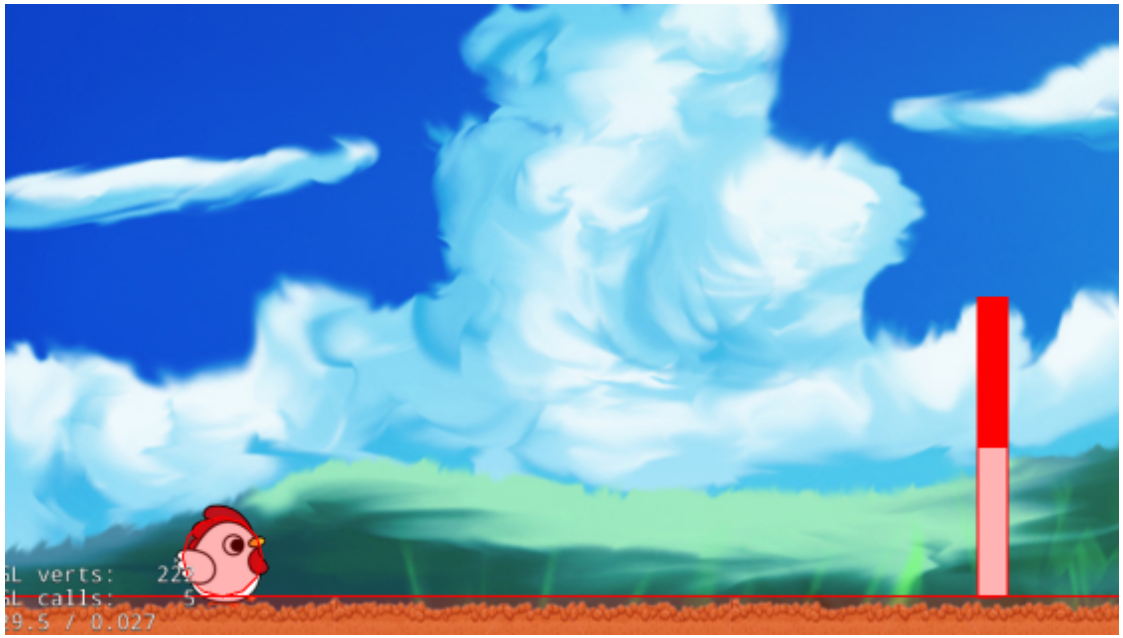
AngryChickenをxcodeで実行。



まずは必要な処理を考えてみる

①一定時間オブジェクトが動いていない
　という判定が必要
②満たしていたら初期状態へ戻す

①一定時間オブジェクトが動いていないという判定方法

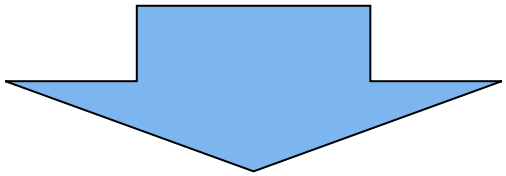


つまりチキンと右側のブロックが静止状態になったらリセットする。

ひとまず今回はチキンを飛ばし、チキンが静止状態に戻ったらリセットするようにしてみる。

「チキンが静止している」＝「どの方向にも力が働いていない」
ということ

チキンと紐づいている剛体クラスには、物理演算処理で
必要な情報が保持されているので、
「現在どれだけ力が働いているか」という値を取得できるメソッド
がありそうな予感



http://www.cocos2d-x.org/reference/native-cpp/V3.10/d7/d7b/classcocos2d_1_1_physics_body.html

PhysicsBody::getVelocity …
velocityは日本語で速力。この関数は方向ベクトルを返すvec2
これがわかればベクトルの長さはVec2::lengthでもとまる。

というわけで、getVelocity()が返す値が0ならば
動いていないということ！

ただし！

本当に0だとガチで止まるまで終わらなくなる。
つまり0.001ずつ動いている（見た目的に静止しているように見える）でもやり直しされなくなるので、
あそびをいれることにする。
（１以下くらいにしとく）

こうなる。

```
void MainGame::update(float delta)
{
    auto* draw = dynamic_cast<DrawNode*>(this->getChildByTag(PULL_LINE_OBJTAG));
    if ( draw != nullptr )
    {//存在した場合＝ひっぱり中なのでひっぱりラインを更新
        draw->clear();
        auto* charSprite = (Sprite*)this->getChildByTag(CHAR_OBJTAG);
        auto charSpritePos = charSprite->getPosition();

        //長さを求める
        auto vec = _currentTouchPoint-charSpritePos;
        float distance = _currentTouchPoint.getDistance(charSpritePos);
        if ( distance > 130 )
        {
            float tDis = 130/distance;
            vec*=tDis;
            _currentTouchPoint = charSpritePos+vec;
        }

        draw->drawSegment(Vec2(charSpritePos.x, charSpritePos.y),
                          Vec2(_currentTouchPoint.x, _currentTouchPoint.y), 2.0f, Color4F(.5f, 1.0f, .7f, 1.0f));
    } else
    {
        // 動作が止まっているか判定する
        auto* charSprite = (Sprite*)this->getChildByTag(CHAR_OBJTAG);
        auto velocity = charSprite->getPhysicsBody()->getVelocity();
        if ( velocity.length() < 1 && this->_touchEndedFlag == true )
        {// リセットする処理をここに

        }
    }
}
```

引っ張っていないとき以外にチキンの速力を1以下になるか監視。しかし！それだけだと、開始した瞬間に即やりなおしとなってしまい永久にゲームができなくなるので「飛ばした」というフラグ（_touchEndedFlag）を１つ用意してそれも条件として付与。なのでフラグもしっかりと宣言・初期化しよう。

②満たしていたら初期状態へ戻す

やり直しにはチキンの位置などもろもろを初期化する必要がある。今はinitに初期化処理が入っているが、この中には「ゲーム初回開始時の初期化」と「やりなおしの初期化」が混同している。
つまりイベントハンドラの登録などの起動時に1回でよい処理と、起動時かつやり直されるたびに何度も呼ぶ処理（例えばチキンの座標の初期化）などがありこれらを別メソッドに分けてやる。そしてやり直し時には何度も呼ぶ処理のメソッドを呼び出せるようにしてやればよい。

initメソッドから初回・やり直しのたびに呼びだすべき初期化処理を抜き出し別のプライベートメソッドに。(_commonInit)

```
void MainGame::_commonInit()
{
    // 初期化
    this->_touchEndedFlag = false;  // 0218

    auto winSize = Director::getInstance()->getWinSize();

    // 一端全ての子供をさよならし、全てのアクションを停止
    this->removeAllChildren();
    this->stopAllActions();

    ////////////////////////////
    //背景の読み込み
    auto* bg = Sprite::create("bg.png");
    auto bgSize = bg->getContentSize();
    bg->setPosition(Point(bgSize.width/2, winSize.height / 2));
    this->addChild(bg);

    ////////////////////////////
    // 地面を配置
    auto* floor = Sprite::create("ground.png");
    floor->setPosition(Point(bgSize.width/2 , floor->getContentSize().height / 2));

    auto material = PHYSICSBODY_MATERIAL_DEFAULT;
    auto* floorPb = PhysicsBody::createBox(floor->getContentSize(), material);
    floorPb->setDynamic(false);
    floorPb->setContactTestBitmask(true);
    floor->setPhysicsBody(floorPb);
    this->addChild(floor);

・・・下に続く・・・
```

```
//////////////////////////////
    // キャラクター
    auto* character = Sprite::create("chicken.png");
    character->setPosition(Point(winSize.width / 5 , winSize.height / 2));

    material = PHYSICSBODY_MATERIAL_DEFAULT;
    auto* charaPb = PhysicsBody::createCircle(40, material);
    charaPb->setMass(50.0f);          // 重さを指定（ここが無いと後で飛ばせなくなる）
    charaPb->setContactTestBitmask(true);
    charaPb->setAngularDamping(1.0);    // 回転の減衰をするように！ 0218
    character->setPhysicsBody(charaPb);
    character->setTag(CHAR_OBJTAG);
    this->addChild(character);

    //////////////////////////////
    // ブロック生成
    _createBlocks();

    //////////////////////////////
    // 画面スクロール登録
    auto size = bg->getContentSize();
    this->runAction(Follow::create(character,Rect(0,0,bgSize.width,winSize.height)));
}
```

ちなみに赤文字部分は、やり直しされた時にはすでにチキンなどが存在しているため、いったん全削除する処理を入れてやる。そうしないとエラーで落ちるか、メモリリークになる。

メモリリークとは？

プログラミングにおけるバグの一種。プログラムが確保したメモリの一部、または全部を解放するのを忘れ、確保したままになってしまうことを言う。プログラマによる単純なミスやプログラムの論理的欠陥によって発生することが多い。

どハマリポイント！

青文字部分は回転力の減衰を有効化している。なんと回転は摩擦などを設定しても影響がないという驚愕の事実がここで判明して、これで3時間はまりました。というわけで、回転力を減衰させたい場合は剛体毎に青文字の設定をしましょう。

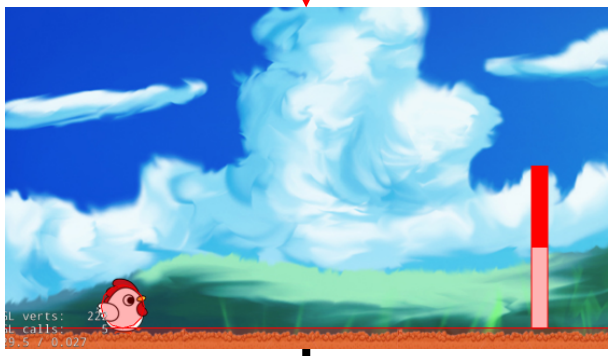
そしてinitはこんな感じになる。

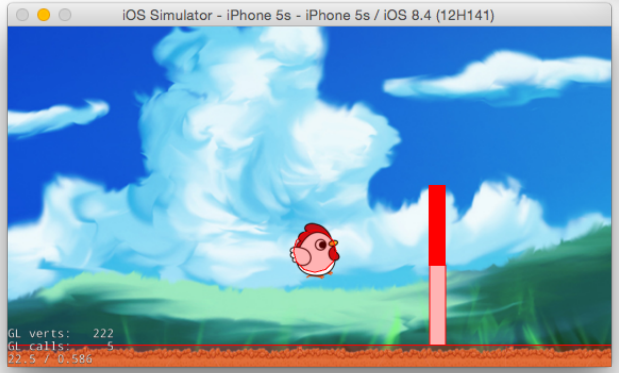
```
bool MainGame::init()
{
    //////////////////////////////
    // 1. super init first
    if ( !Layer::init() )
    {
        return false;
    }

    // まいフレーム呼ばれるupdateメソッドを有効か
    this->scheduleUpdate();

    //////////////////////////////
    // Touchイベント用
    auto eventDispatcher = Director::getInstance()->getEventDispatcher();
    auto listener = EventListenerTouchOneByOne::create();
    listener->onTouchBegan = CC_CALLBACK_2(MainGame::_onTouchBegan, this);
    listener->onTouchMoved = CC_CALLBACK_2(MainGame::_onTouchMoved, this);
    listener->onTouchEnded = CC_CALLBACK_2(MainGame::_onTouchEnded, this);
    listener->onTouchCancelled = CC_CALLBACK_2(MainGame::_onTouchCancelled, this);
    eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);

    //////////////////////////////
    // 物理演算衝突検知メソッド登録 11.22
    auto contactListener = EventListenerPhysicsContact::create();
    contactListener->onContactBegin = CC_CALLBACK_1(MainGame::_onContactBegin, this);
    contactListener->onContactPreSolve = CC_CALLBACK_2(MainGame::_onContactPreSolve, this);
    contactListener->onContactPostSolve = CC_CALLBACK_2(MainGame::_onContactPostSolve, this);
    contactListener->onContactSeparate = CC_CALLBACK_1(MainGame::_onContactSeparate, this);
    this->getEventDispatcher()->addEventListenerWithFixedPriority(contactListener, 10);

    _commonInit();


    return true;
}
```

実行してみよう！

こうなる！

チキンが概ね静止した
ら即リセット





まとめ

回転力の減衰を有効かするには
Physics::setAngularDampingが必須。

物体の速力を得るにはPhysics::getVelosityを
使おう。

次回は物理演算Chipmunk
軌跡の点線を入れてみる編

ここまでのソースプログラムはここ

http://monolizm.com/sab/src/AngryChicken16.zip

ご清聴ありがとうございました。